P3-14 N94-35072

SPACE/GROUND SYSTEMS AS COOPERATING AGENTS

T. J. Grant

BSO/Aerospace & Systems b.v.
P. O. Box 1444
3430 BK Nieuwegein
The Netherlands

Tel: +31-3402 88888
Fax: +31-3402 60577
Email: tigr@bsonwg.bso.nl

ABSTRACT

Within NASA and the European Space Agency (ESA) it is agreed that autonomy is an important goal for the design of future spacecraft, and that this requires on-board Artificial Intelligence. NASA emphasises deep space and planetary rover missions, while ESA considers on-board autonomy as an enabling technology for missions that must cope with imperfect communications. ESA's attention is on the space/ground system.

A major issue is the optimal distribution of intelligent functions within the space/ground system. This paper describes the Multi-Agent Architecture for Space/Ground Systems (MAASGS) which would enable this issue to be investigated. A MAASGS agent may model a complete spacecraft, a spacecraft subsystem or payload, a ground segment, a Spacecraft Control System, a human operator, or an environment. The MAASGS architecture has evolved through a series of prototypes. The paper recommends that the MAASGS architecture should be implemented in the operational Dutch Utilisation Centre.

INTRODUCTION

Within NASA and the European Space Agency (ESA) it is agreed that autonomy is an important goal for the design of future spacecraft, and that this requires on-board Artificial Intelligence. NASA's emphasis has been on deep space and planetary rover missions. ESA is considering greater on-board autonomy as a potential enabling technology for missions that must cope with communication delays or interruptions, as well as a way of reducing spacecraft operations costs. A series of ESA studies has resulted in the development of the Standard Generic Approach to Spacecraft Autonomy and Automation (SGASAA) concept.

Until recently, the emphasis has been on the space segment. ESA's attention is now turning to the complete system comprising both the space and ground segments: the *space/ground system*. A major issue is the optimal distribution of intelligent functions such that the space/ground system design results in a clearly quantifiable reduction in operational costs, without other adverse effects (e.g., on spacecraft reliability). Potential applications are foreseen in the ground-based Command and Control (C^2) of spacecraft which are subject to delays or interruptions in communication, e.g. deep space missions and missions partly visible from ground stations.

This paper describes the Multi-Agent Architecture for Space/Ground Systems (MAASGS) which enables the issue to be investigated. A MAASGS agent may model a complete spacecraft, a spacecraft subsystem or payload, a ground segment, a Spacecraft Control System, a human operator, or an environment. The architecture - developed for the Dutch Utilisation Centre (DUC) (Pronk, Koopman & de Hoop, 1992) - is based on Multi-Agent Systems (MAS) techniques. A MAASGS agent may model a complete spacecraft, a spacecraft subsystem or payload, a ground segment, a Spacecraft Control System, a human operator, or an environment. The MAASGS architecture has evolved under company and Dutch national investment through a series of prototypes. The paper concludes that the architecture is now mature, and recommends that it be implemented for use in the operational DUC.

There are five sections in this paper. Section 2 outlines the SGASAA concept. Section 3 motivates the use of MAS techniques. Section 4 describes the MAASGS architecture, including its evolution and associated development methodology. Finally, Section 5 draws conclusions and makes recommendations.

SGASAA CONCEPT

Defining Autonomy

Spacecraft autonomy can be loosely defined as the ability of a spacecraft to be largely or wholly independent of ground control (Pidgeon, Seaton, Howard and Peters, 1992). More precise definitions are mission-dependent. For scientific and communications satellites, the main drivers for autonomy are:

- Short and infrequent periods of ground station contact mean that there is little visibility of on-board events. Consequently, there is little opportunity for ground-based control to influence on-board events. The spacecraft must perform basic monitoring and control.

- Long transmission delays mean that the mission would not be practicable without some degree of autonomy.

- The need to maximise the mission product in the event of an internal or external event (e.g. on-board failure or change in its environment) means that the reaction time should be kept as short as possible. Autonomy reduces the need for the spacecraft to refer to the ground segment for a decision.

- Long duration missions where operations costs could be significantly reduced.

Autonomous Functionalities

A number of studies (Devita and Turner, 1984), (Doxiadis, 1988), (Drabble, 1991), (Elfving and Kirchhoff, 1991) have been conducted for various agencies to investigate approaches to spacecraft autonomy. ESA's studies, begun in the early 1980s, culminated in the SGASAA concept (Berger, Cornet, Cellier, Riou, Sotta and Thibaut, 1984). By the beginning of the 1990s, ESA had progressed to validating the SGASAA concept, in the Spacecraft Autonomy Concept Validation (SACV) study (Pidgeon; Seaton, Howard and Peters, 1992).

For scientific satellites, autonomy is viewed as replacing (or supplementing) ground-based operator functions with on-board functions. The SACV study listed the foreseen on-board functionality as:

- Execution, updating and rescheduling of a *Master Schedule*, which is a set of high-level, time-tagged, goal-oriented commands stored on-board. Rescheduling would take into account the commands' resource requirements, the availability of on-board resources, the dependencies between commands, and environmental and timing constraints.

- Fault diagnosis would be performed on-board. The autonomous spacecraft would attempt to recover from a failure, while ensuring the spacecraft's safety and minimising the loss of the mission product. Fault diagnosis could only cater for foreseen failure modes. Unforeseen failures would have to result in the spacecraft adopting a safe mode to await ground intervention.

For reasons which are unclear, the SACV study omitted a third possible on-board functionality: goal-oriented planning. Goal-oriented planning was always seen as having an equal priority with other functionalities (e.g. see (Berger, Cornet, Cellier, Riou, Sotta and Thibaut, 1984), Volume 1, Figure 5.2/4). Therefore, this paper assumes that the on-board functionality must include:

- Goal-oriented planning (and re-planning), which must:
  - Take the (re-)planning activity into account.
  - Be interruptible.
  - Be able to generate alternative plans for the same requirements.

The Concept

A common thread amongst the spacecraft autonomy studies has been the adoption of a hierarchically-based On-Board Management Systems (OBMS). The OBMS

consists of a high-level On-Board Mission Manager (OBMM) together with various subordinate subsystem and payload managers (generically termed Sub-System Managers (SSMs)). The OBMM monitors, coordinates and controls the SSMs, and each SSM monitors, coordinates and controls a subsystem or a payload. The OBMS is supported by a distributed on-board communications architecture with the managers communicating via a LAN or databus, and each subsystem and payload being connected to its SSM via a subassembly LAN. The SSM effectively acts as a bridge between the subassembly LAN and the spacecraft LAN. From outside the subsystem or payload, the SSM appears to 'wrap' the subsystem or payload with additional functionalities.

The SGASAA concept adopted a distributed hierarchy on the grounds that decision-making should be devolved to the lowest possible level in the hierarchy. Figure 1 depicts the conceptual SGASAA architecture. The spacecraft consists of a set of "intelligent" subsystems and payloads, each of which has the capability to interpret Telecommand (TC) packets and to generate Telemetry (TM) packets. Packetised TM/TC is a prerequisite for the SGASAA approach.

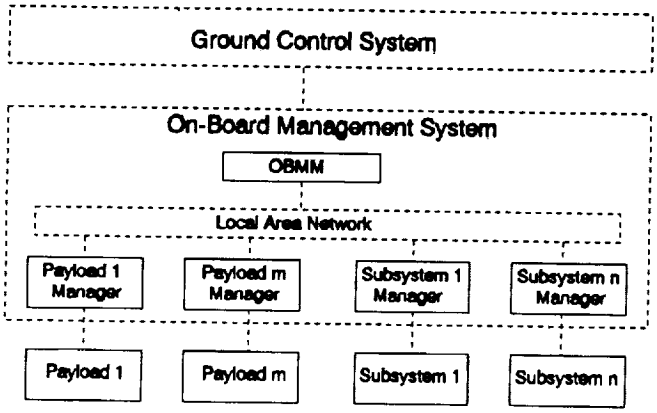

Figure 1: SGASAA Architecture.

The SGASAA approach relies on the concept of a series of layered mission plans. Low-level plans are generated from the plan above by adding detail. For example, the *Long Term Operations Plan* (LTOP), which defines the objectives for an entire mission or mission phase, can be broken down into a series of components (*Links*) which are uplinked to the spacecraft and executed on-board. Links describe actions which can be achieved at system level by a combination of activities at subsystem level. The *Short Term Operations Plan* (STOP), covering a period of several days, consists of a set of links with coarse parameters. These are rescheduled with precise parameters, reflecting the current on-board state, and broken down into a sequence of blocks. The *Executable Operations Plan* (EOP) contains blocks of actions for a single subsystem or payload, usually in the form of macro-commands which are expanded on-board. The *Elementary Commands* (EC) are time-tagged commands contained within an EOP, each normally affecting only a single element of the subsystem, e.g. switching a heater on.

Plans are validated and optimised at each level, refining the plan from a coarse LTOP to detailed ECs. The SGASAA concept also allows for direct commanding of subsystems, bypassing the OBMM. A limitation of the SGASAA plan hierarchy is that higher-level plans must contain information about the lower-level plans, such as resource usage, duration, dependencies, etc. This means that planning is necessarily an iterative process, with lower-level plans providing feedback to higher-level plans.

The OBMM and the Subsystem and Payload Managers have prescribed roles (see (Pidgeon, Seaton, Howard and Peters, 1992), section 2.3). Comparison of these roles shows that each SSM has the same functionality as the OBMM, albeit for a more detailed subset of the spacecraft (i.e. the subsystem or payload for which the SSM is responsible). The common functionalities are:

- Distribution and execution of TCs.
- Generation of TMs.
- Fault diagnosis.
- Failure recovery.
- Localised planning.
- Self-checking.

The SGASAA concept defines three modes of operations:

- Routine Mode, in which nominal and expected tasks are executed.
- Crisis mode, which is the handling of unexpected events on-board the spacecraft or due to external influences.

Check-out Mode, in which the spacecraft is placed in a configuration which allows hardware and/or software to be tested.

MOTIVATION FOR MULTI-AGENT SYSTEMS

Multi-Agent Systems

Distributed Artificial Intelligence is defined as "the subfield of AI concerned with concurrency in AI computations" (Bond and Gasser, 1988, p.3). Bond and Gasser divide the world of DAI into three arenas: Distributed Problem Solving, Multi-Agent Systems, and Parallel AI. In this paper, we are concerned with Multi-Agent Systems (MASs), i.e. "with coordinating intelligent behaviour among a collection of (possibly pre-existing) autonomous intelligent 'agents', which can coordinate their knowledge, goals, skills, and plans jointly to take action or to solve problems" (*ibid.*, p. 3). Typical intelligent behaviours are to generate plans and schedules, to react appropriately to situations (including diagnosing and recovering from failure), and to learn. The agents may be working towards a single, global goal, or towards separate, individual goals that can conflict. Crucially, "they must ... reason about the *processes of coordination among the agents*" (Bond and Gasser, 1988, p.3, italics in original). The task of coordination can be difficult, because there may be situations where there is no global control, no globally consistent knowledge, no globally shared goals, and/or no global success criteria. Reviews of MAS techniques and trends may be found in (Castillo-Hern and Wilk, 1988), (Grant, 1992), and (Chaib-Draa, Moulin, Mandiau and Millot, 1992).

There is no consensus definition of an agent. Bond and Gasser skirt around the issue; they rely on a simple and intuitive notion of an agent as a computation process with a single locus of control and/or "intention" (*ibid.*, p.3, footnote 1). For the purposes of this paper, an *agent* will be defined as a software entity with autonomous processing capabilities and a private database, which acts on its environment on the basis of information it receives from the environment.

Motivations for Using MAS Techniques

Huhns (1987) lists five primary reasons why one would want to use MAS techniques:

- MASs can provide insights and understanding about interactions among humans, who organise themselves into various groups, committees, and societies to solve problems.
- MAS techniques can provide the means for interconnecting multiple expert systems that have different, but possibly overlapping, areas of expertise. This permits the solution of problems whose domains lies outside the area of expertise of any one expert system.
- MASs can potentially solve problems that are too large for a centralised system because of resource limitations (eg bandwidths, computing speeds, and reliability) induced by technology.
- MASs can potentially provide a solution to a current limitation of knowledge engineering: the use of only one expert. If there are several experts or several non-experts whose ability can be combined to give expert-level behaviour, there is no established way to use them successfully.
- MAS techniques are the most appropriate solution when the problem itself is inherently distributed, as in distributed sensor networks and distributed information retrieval.

Clearly, the last reason is the prime motivation for applying MAS techniques to space/ground systems.

Huhns (1987) also listed the following advantages for system development:

- Partitioning the software system into agents reduces the complexity, resulting in a system that is easier to develop, test, and maintain.
- The software subsystems (i.e., agents) can operate in parallel.
- The software system can be designed - using the *functionally accurate* approach (Lesser and Corkill, 1981) - to continue to operate even if part of it fails.
- It is easier to find experts in narrow domains.

MAS Issues

The following issues are relevant to this paper:

- Structuring the functionalities internal to an agent. A wealth of differing agent structures exists in the MAS and C^3I literatures. The MAASGS architecture has evolved an agent structure from object-oriented systems by adding concepts from C^3I theory and then by specialising this for spacecraft operational control.

- Representing the agent's knowledge of its environment. Investigation of possible ways of representing the agent's knowledge of its environment is a major sub-field of MAS research. Representations vary from the agent-attribute-value model to logics of belief which are modelled on human psychology. We use the simple agent-attribute-value model, with the attributes being typed according to the functionality which operates on them. For example, the attributes might be rules, Horn clauses, planning operators, or constraints, as well as datatypes such as booleans, integers, reals, strings, etc.

- Enabling agents to communicate with one another. Chaib-Draa, Moulin, Mandiau and Millot (1992) identifies solutions to inter-agent communications ranging from *no communication*, through *primitive communication, plan and information passing*, information exchange via a *blackboard*, *message-passing*, to *high-level communication*. The MAASGS architecture adopts the message-passing model because this models closely the packetised TM/TC used in modern spacecraft, and can be readily implemented using the message-passing model employed in object-oriented programming languages such as Smalltalk, C++, CLOS and Eiffel.

- Enabling agents to coordinate their actions. Agents must coordinate their distributed resources, which may be physical or computational. The most appropriate coordination technique depends on the distribution of the shared resources and on the local autonomy of agents, which may have disparate goals, knowledge and reasoning processes. Generally, DAI researchers use the *negotiation* process to coordinate a group of agents. There are various definitions for negotiation. We adopt Bussmann and Müller's (1993) definition of negotiation as "the communication process of a group of agents in order to reach a mutually accepted agreement on some matter". A variant is *arbitration*, in which the group of agents appeal to an impartial agent to reach the agreement. In DAI, negotiation is often implemented as the Contract Net Protocol (Davis and Smith, 1983), in which an agent needing help decomposes the problem into subproblems, announces the opportunity to solve the subproblems to the group, collects bids for their solution from group members, and awards the subproblems to the most suitable bidders. The MAASGS architecture can accommodate a range of coordination protocols, including an arbitration protocol which supports inter-agent learning (Grant and Lenting, 1993).

- Modelling domains by means of agents. Borrowing from object-oriented simulation, we follow the fundamental principle of modelling each real-world object - whether or not it has any intelligent functionality - as an agent. There are two ways to model domains in this way: agents may be *specialists* or they may be *generalists*. Specialist agents have functionality that is specific to the role of the real-world object being modelled, e.g., transforming X-rays into data, calculating spacecraft orbits, and so on. By contrast, generalist agents all have the same generic functionalities, e.g., rule-based inference, goal-oriented planning, constraint-based scheduling, and so on. The MAASGS architecture employs generalist agents. Examples of the generic functionalities in the MAASGS architecture are receipt of TCs, generation of TMs, monitoring other agents' status, diagnosis, selection of procedures, goal-oriented planning, scheduling, etc. There is a small set of agent-classes, derived from Grant's (1992a) abstraction hierarchy. The agent-class which models the non-intelligent domain objects, such as payload components,

implements only receipt of TCs, generation of TMs, and internal computation. Another agent-class models intelligent domain objects, such as SSMs.

Organising agents to represent distributed systems. Elaborate schemes have been devised to represent organisations of agents. We have adopted the simple idea that an agent can be decomposed into more primitive agents. Despite its anthropomorphic title in the DAI literature - where it is known as Minsky's (1986) "Society of Minds" concept - the idea of decomposition is to be found in any industrial-strength software analysis or design method, e.g. SADT and dataflow diagramming. Domain decomposition hierarchies are usually easy to find. For example, the very first sentence in an ESA Bulletin article on the use of spacecraft simulators at ESOC (Gujer and Jabs, 1991) states (p. 41):

"A satellite mission can be considered in its simplest form to consist of a space segment, a ground segment and a user community ... The ground segment for an ESA mission ... includes: a set of ground stations, ... a communications network, ... the Operations Control Centre (OCC), ... payload data-processing facilities ..."

The same article later states (p. 46):

"Figure 6 shows the layout of a typical spacecraft model as implemented in most simulators. It closely reflects the standard decomposition of a spacecraft into subsystems."

In the MAASGS architecture, decomposition hierarchies are modelled by enabling any agent to have zero or one *superior* agents and zero or more *subordinate* agents. The superior represents the assembly of which the agent is a part, and the subordinates represent the component parts of the agent. This approach implies that each node in the decomposition hierarchy is modelled as an agent, and not just the leaf-nodes.

THE MAASGS ARCHITECTURE

Evolution

The MAASGS architecture has evolved by specialising the generic agent structure. The first step was to incorporate classic C^3I features, based on Wohl's Stimulus-Hypothesis-Option-Response (SHOR) model of decision-making (Wohl, 1981). This resulted in the Message-Based Architecture testbed (Grant, 1991), developed as a private venture. The testbed was designed primarily as a "test harness" for an inductive learning algorithm. The reactive and generative planning functionalities were deliberately designed to be the minimum necessary to close the loop from the inductive learning algorithm's output back to its input. The testbed successfully demonstrated *learning-by-doing* (Anzai and Simon, 1979).

The second step was a paper study of an agent structure suited to the Columbus User Support Organisation (USO), based on the lessons learned in developing and using the Message-Based Architecture testbed. This study was a part of BSO/Aerospace & Systems' "DUC Preparation" (DUCPREP) project. The DUCPREP project was funded by company and Dutch national investment and performed in informal cooperation with a number of other Dutch companies. The agent structure proposed for the DUC was documented in (Grant, 1992a).

In the third step, the internal functionalities of a Message-Based Architecture agent were extracted and enhanced. The resulting DUC Activity Scheduling System (DUC-ASS) is a single-agent software system capable of integrating the support of payload design, planning, scheduling, and control (Grant, 1992b). Prototyping of the DUC-ASS was performed under BSO/Aerospace & Systems' "MILDS" project, also funded by company and national investment. The MILDS project formed an element of the larger "DUC-Pilot" project performed by a Dutch consortium. Under the DUC-Pilot project, an interface was defined (Grant and Tusveld, 1992) for coupling the DUC-ASS to a diagnostic system which used model-based reasoning techniques. Current DUC-related developments (Pronk, Visser and Sijmonsma, 1993) centre on linking the pilot DUC to ESTEC's Crew Work Station testbed for Mission Simulation purposes.

The MAASGS architecture uses the DUC-ASS agent structure, enhanced to incorporate the scheduling and model-based diagnosis functionalities. Although the MAASGS architecture has not yet been implemented in full, the key functionalities have all been implemented. The interfaces between them have been defined to varying levels of detail. Two related issues have been addressed by exploratory prototyping: recognising objects during learning (Grant, van Meenen and Stroobach, 1992), and the modeller's graphical user interface (Grant, 1993a). In addition, the Message-Based Architecture testbed has been recently enhanced to enable agents to exchange learned knowledge, i.e. they can also learn-by-being-told (Grant, 1993b).

Agent Structure

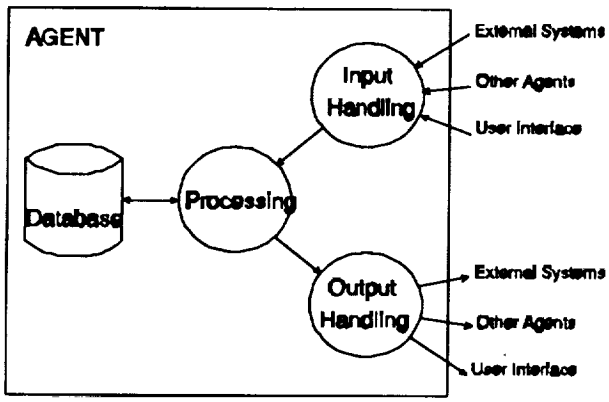

Figure 2: Generic Structure of an Agent.

There are four components in a generic agent (see Figure 2). *Input Handling* receives and filters the incoming messages from other agents, from a user interface, and from external systems. *Processing* operates on the filtered incoming messages, retrieving and storing information in the agent's private *Database*, and generating messages to send. *Output Handling* formats and despatches the outgoing messages to other agents, to the user interface, and to external systems.

Such an agent is a specialisation of an object in object-oriented systems. An agent has a unique name, autonomous processing capabilities (cf. methods), and a private database (cf. attributes and their values), and exchanges information with its environment (cf. message-passing). Some MASs also have agent classes and inheritance. MASs have functionality that extends beyond that of object-oriented systems. In particular, agents in MASs are *intelligent agents*. Typical intelligent behaviours are to react appropriately to situations, to generate plans, and to learn. These behaviours can be best modelled using AI techniques, such as expert systems, knowledge-based planning (Georgeff, 1987), and machine learning (Michalski, Carbonell and Mitchell, 1983). We distinguish agents from objects by requiring that an agent minimally includes the abilities to:

- Model its own state and behaviour, and
- Decide whether or not to accept a new state its environment attempts to impose on it.

Such an agent is termed a *non-intentional* agent (Grant, 1992). *Intentionality* means to have attitudes towards other agents (Searle, 1980), such as intentions, goals, desires, or beliefs. Any agent which generates instructions, forms plans, or learns about other agents is necessarily intentional.

An abstraction hierarchy of agents may be built on the minimal set of abilities. Grant (1992) proposes an abstraction hierarchy in which an *intentional* agent also has the abilities to:

- Model its own goals,
- Model other (non-intentional) agents, and
- Manage a negotiation or arbitration process between other agents.

At the very least, intentional agents are aware of the existence of other agents in their environment. Following the precedent set by the MACE testbed (Gasser, Braganza and Herman, 1987), the other agents are usually known as the agent's *acquaintances*. In many MASs, agents also know about their acquaintances' behaviours, i.e., their *capabilities*.

Agent-Based Simulation

Application domains may be modelled as collections of agents. As in object-oriented simulation, a set of entities must be provided which the modeller can instantiate to represent the domain. There are two fundamental set-elements in object-oriented simulations: *object-classes* (cf. Smalltalk's Class object-class) and *messages* (cf. Smalltalk's Message object-class). In agent-based simulation, the equivalent entities are *agent-classes* and *messages*. The distinction between

objects and agents implies that additional entities are needed to represent the agent structure. Precisely what additional entities are provided depends on the simulation development environment designer. Our experience shows that a suitable set of additional entities should include *agent structuring*, *inter-agent message-handling*, *inter-system interfacing*, *user interface*, and *agent organisation* entities. More details are in (Grant, 1993a).

In addition, there must be a domain-independent Simulation Development Environment (SDE), comprising a simulation executive, a set of tools, a user interface, and, optionally, interfaces to external systems. A database management system may also be provided where the agents in the simulation model are not persistent. The SDE may itself be implemented as a second collection of agents. Issues concerning the SDE are outside the scope of this paper. The wider issues concerning how AI and simulation techniques may complement one another are covered by Widman, Loparo and Nielsen (1989).

Adding C³I Features

The SHOR model describes a data-driven or reactive approach to problem-solving and decision-making. The model identifies four information-handling processes. *Stimulus* involves the processing of raw data received from the decision-maker's environment via sensors. Processing includes searching for data, scanning or sampling it, reducing, compressing, and aggregating the scanned/sampled data, and detecting, recognising and confirming events signalled by the data. The decision-maker interacts with the environment, eg by directing sensors. *Hypothesis* involves the generation and evaluation of hypotheses concerning the environment's state-of-affairs, based on the outputs of Stimulus. Processing includes data association and correlation, state and parameter estimation, hypothesis generation, situation assessment, and decision state estimation. The decision-maker is essentially passive to the external environment while he/she focuses on the analysis task. *Option* concerns the generation, planning and evaluation of alternative options for the decision-maker's response to the estimated decision state. *Response* concerns the execution of the selected response. Execution involves the issue of information to the decision-maker's environment, either by physical action or by communicative action.

Various authors view Wohl's Stimulus and Hypothesis in terms of the data processing techniques employed. Event detection, recognition, and confirmation through to decision state estimation are grouped together as *data fusion*, defined (Waltz & Llinas, 1990, p. 1) as:

> "A multi-level, multi-faceted process dealing with the detection, association, correlation, estimation and combination of data and information from multiple sources to achieve refined state and identity estimation, and complete assessments of situation ...".

Thus, data fusion maps onto the latter part of Stimulus, combined with Hypothesis. In Wohl's Hypothesis process, the decision-maker associates the events recognised during Stimulus processing with objects in the environment. The states and other parameters of these objects can then be estimated.

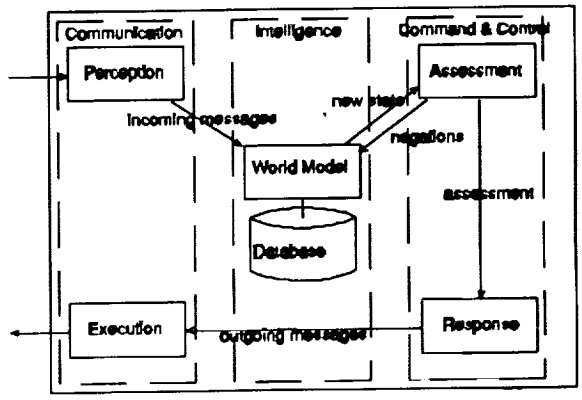

Figure 3: Agent Structure Enhanced with C³I Features.

Enhancing the generic agent structure results in the structure shown in Figure 3. Input Handling (cf. Stimulus) becomes the Perception module, and output processing becomes Execution. Processing is now divided into the Assessment and Response modules (cf. Hypothesis and Option), and World Model encompasses Database. Figure 3 shows the modules grouped by the military terms "Communication", "Intelligence", and "Command and Control".

The C³I-enhanced structure works according to the "do-as-little-work-as-possible" principle. For example, if the Perception module filters out an incoming message as having nothing to do with the agent, processing stops

at that point. Similarly, the Assessment module, which uses constraint-based techniques to check the consistency of the World Model after the incoming information has been added to it, can decide to terminate processing if the incoming information is consistent with what the agent already knows. Only if an inconsistency or conflict is found does the Assessment module trigger the Response module.

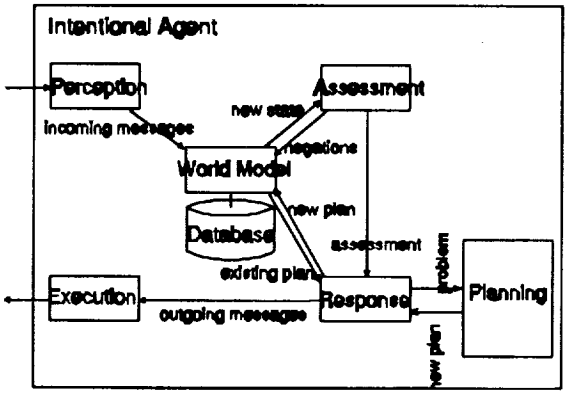

Figure 4: Structure of Intentional Agent.

The agent structure shown in Figure 3 has the capability of a non-intentional agent. It can react to events, but it cannot plan ahead. In AI terms, it is limited to forward-chaining (or data-driven) reasoning. The danger is that such an agent can get into a situation from where it is impossible to reach its goals: a "cul-de-sac" world-state. To obtain an intentional agent, the agent structure must be further enhanced with planning functionality (see Figure 4). The Planning module can be seen as providing a planning service to the Response module. Other intentional functionalities, such as scheduling and learning, can be added as further services to the Response module.

MAASGS Architecture

The MAASGS agent structure, shown in Figure 5, has clear correspondences to the structure of an intentional agent in the Message-Based Architecture test-bed. The Monitoring module, together with the Status Database, is equivalent to the Message-Based Architecture's Perception and World Model modules. The Detection module performs situation assessment. Response selection is performed by the Isolation, Diagnosis, and Recovery modules. The Real-Time Replanning and Scheduling module replaces the Planning Module. The Execution and Predictive Payload Simulator (PLS) modules perform the functions of the Message-Based Architecture's Execution module.

The Detection, Isolation, Diagnosis and Recovery modules are grouped together as the ADIR assembly, where the "A" stands for "Anomaly". Normally, such a grouping of functionalities would be termed "FDIR", where the "F" stands for "Fault". In the MAASGS architecture, the functionalities are generalised to encompass anomalous situations. An *anomaly* exists whenever the telemetry indicates that a parameter has a value which either falls outside its alarm or warning levels or is unplanned or unexpected. Unplanned values may still be beneficial, i.e., serendipitous. Therefore, this functional group must not be regarded as FDIR, until the presence of a fault has been confirmed.

Limitations in the available funding have meant that the MAASGS concept has not yet been implemented fully. A number of prototypes of MAASGS modules exist. For example, the ADIR group of modules has been developed fully. Prototype PLSs exist, but have not been tailored for prediction and anomaly detection. The DUC-ASS application can be seen as a prototype non-real-time Replanning and Scheduling module. User interface issues have been partly explored in the agent-based Application Data Source Simulation Tool (ADSST) (Grant, 1993a). Between them, these prototypes have covered the MAASGS functionalities. It now remains to integrate them, ideally using the emerging international standards for knowledge representation (Grant and Poulter, 1993).

Modelling Methodology

The MAASGS modelling methodology starts with domain decomposition. This can be done top-down, bottom-up, or middle-out. The second step is to define the internal database and processing functionality of each agent. Inter-agent connectivity is defined in the third step in terms of links and switches. The agents are initialised in the fourth step. In the fifth step, simulation scenarios are defined. In several of the MAASGS prototypes, scenarios can be defined by direct manipulation of the agents, with the system capturing the user's manipulations and compiling them as a scenario. A chosen simulation scenario is run in the sixth step, and the results are evaluated in step seven.

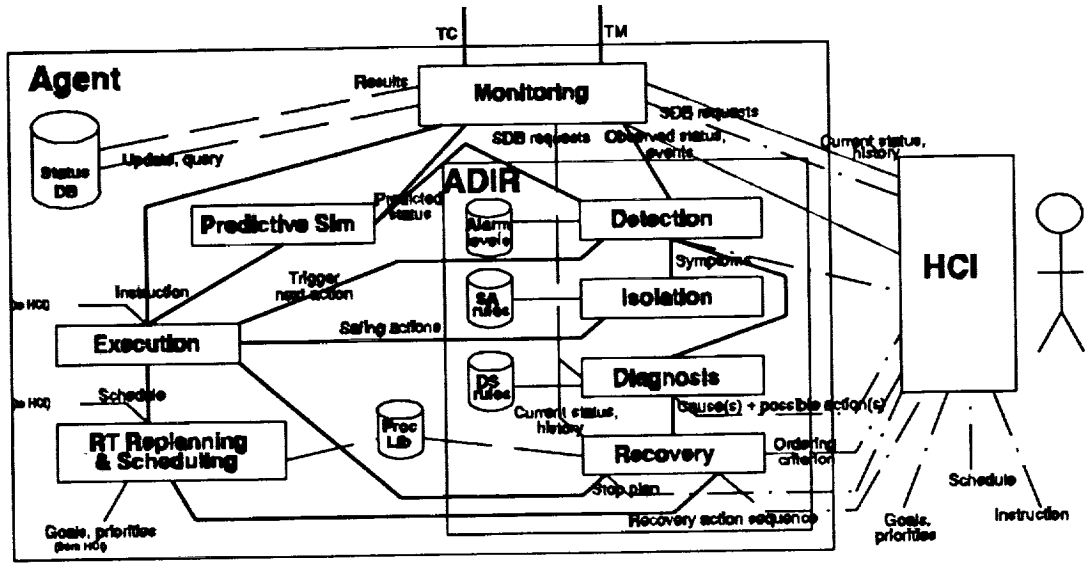

Figure 5: MAASGS Agent Structure.

Further Work

Conceptually, the MAASGS architecture is mature. It has evolved through iterative enhancement, with each step being tested by implemented systems. All the MAASGS components exist in developed or prototype form. The MAASGS architecture should now be implemented in full.

The first step would be to define all the inter-module interfaces, preferably using a knowledge communication standard. In parallel, the user interface prototyping begun in the ADSST should be extended. The MAASGS implementation would be built up step-by-step, starting with a payload simulator to represent the agent's environment. A suitable sequence for adding the modules would be: HCI, Monitoring, Status Database, Anomaly Detection, Isolation, Execution, Predictive PLS, Diagnosis, Recovery, Real-Time Replanning, and finally Scheduling.

Having built up to the full agent structure, a second phase could begin. From a single agent communicating with a (simulated) payload, a second agent could be introduced. This would be best done by inserting the second agent between the simulated payload and the existing agent, to model the situation in which a User Home Base (i.e., the first agent) is subordinate to a Payload Operations Control Centre (i.e., the second agent). This situation would model the Dutch Utilisation Centre (DUC), which combines payload control at the Dutch national level (i.e., a User Support Operations Centre, in Columbus/Space Station Freedom terminology) with a User Home Base. Additional agents could then be introduced to model further User Home Bases.

Having achieved a hierarchical agent model of the ground segment, attention could then turn to the space segment model. The simulated payload would be replaced by a further agent. The remainder of the space segment would then be introduced as additional agents. A final refinement would be to represent the communications chain linking the space and ground segments also as agents. Any part of the complete agent-based space/ground system model could be modelled to a higher fidelity at any time by

decomposing the part concerned into subordinate agents.

Such an agent-based model of the complete space/ground system could be used in an Operations Research (or Management Science) role. Functionalities of selected agents could be switched on or off to investigate the optimal distribution of functionality. New functionalities, such as agent learning, could be assessed by enhancing the generic agent structure. Different space/ground system architectures could be investigated by altering the connectivity between selected agents. For example, "deputy" agents could be introduced for key roles, such as the OBMM or the Payload Operations Control Centre. Instead of an OBMS hierarchy, a heterarchy of SSMs could be evaluated. Another application would be to model spacecraft constellations by instantiating multiple space segments.

An agent-based model of the space/ground system could also be used for operational purposes. It could be used to evaluate the introduction of additional payloads and Principle Investigators. It could be used to evaluate mission timelines. Finally, by replacing one or more software agents with real payloads, subsystems, control systems, and people, it could be used for verification and training purposes.

CONCLUSIONS AND RECOMMENDATIONS

This paper has described ESA's Standard Generic Approach to Spacecraft Autonomy and Automation (SGASAA). The on-board functionalities have been outlined. The SGASAA architecture has been depicted. Multi-Agent Systems (MAS) have been defined, the motivations for using MAS techniques have been listed, and relevant MAS issues have been discussed. The paper has shown that MAS techniques are the most appropriate solution to modelling space/ground systems, because such systems are inherently distributed.

BSO/Aerospace & Systems' Multi-Agent Architecture for Space/Ground Systems (MAASGS) has been documented. Its evolution has been sketched. The MAASGS agent structure has been detailed. The addition of agent-based simulation and C^3I features has been described. The methodology for using the MAASGS architecture has been outlined. Further work has been identified. The paper concludes that the MAASGS architecture is conceptually mature, and recommends that the architecture should now be implemented in full.

REFERENCES

Anzai, Y., and H. A. Simon. (1979). The Theory of Learning by Doing, Psychological Review, 86, 2, 124-140.

Berger, G., J. Cornet, M. Cellier, L. Riou, J. Sotta, and M. Thibaut. (1984). Standard Generic Approach for Spacecraft Intelligence and Automation: Final Report, ESA Contract number 4869/81/NL/PP, ESA Report CR(P) 1759 Volumes 1 and 2.

Bond, A. H., and L. Gasser, (eds). (1988). Readings in Distributed Artificial Intelligence, Morgan Kaufman, San Mateo, CA, USA.

Bussmann, S., and J. Müller. (1993). A Negotiation Framework for Cooperating Agents, In Deen, S. M. (ed). (1993). 1992 Proceedings of Special Interest Group on Cooperating Knowledge-Based Systems (CKBS-SIG'92), DAKE Centre, University of Keele, UK, 1-17.

Castillo-Hern, L. E., and P. F. Wilk. (1988). Describing DAI Models: A Framework and Examples, Proceedings, Alvey workshop on Multiple Agent Systems, Philips Research Laboratories, Redhill, UK, 14/15 Apr 88.

Chaib-Draa, B., B. Moulin, R. Mandiau, and P. Millot. (1992). Trends in Distributed Artificial Intelligence, Artificial Intelligence Review, 6, 1, 35-66.

Davis, R., and R. G. Smith. (1983). Negotiation as a Metaphor for Distributed Problem Solving, Artificial Intelligence Journal, 20, 63-109.

Devita, E. L., and P. R. Turner. (1984). Autonomous Spacecraft Design Methodology, NASA CR 849323, Jet Propulsion Laboratory, Pasadena, California, USA.

Doxiadis, A. S. (1988). Expert Systems for Spacecraft Autonomy, Ford Aerospace, International Federation of Automatic Control Workshop in Spacecraft Autonomy.

Drabble, B. (1991). Spacecraft Command and Control using Artificial Intelligence, JBIS, 44, 6, 251-254.

Elfving, A., and U. Kirchhoff. (1991). Design Methodology for Space Automation and Robotics Systems, ESA Journal, 15, 149.

Gasser, L., C. Braganza, and N. Herman. (1987). MACE: A Flexible Testbed for Distributed AI Research, chapter 4 in (ed) Huhns, M. (1987). Distributed Artificial Intelligence, Research Notes in Artificial Intelligence, Pitman, London, UK, 119-152.

Georgeff, M. P. (1987). Planning, American Reviews in Computer

Science, 2, 359-400. Also published in (eds) Allen, J., J. Hendler, and A. Tate. (1990). Readings in Planning, Morgan Kaufman, San Mateo, CA, USA, 5-25.

Grant, T. J. (1991). Integrating Reactive Planning, Plan Generation, and Planning Operator Induction in the Message-Based Architecture, Proceedings, 10th UK Planning SIG workshop, Logica Cambridge Ltd, Cambridge, UK, 25/26 April 1991.

Grant, T. J. (1992). A Review of Multi-Agent Techniques, with application to Columbus User Support Organisation, Proceedings, AI and KBS for Space Workshop, ESTEC, 22-24 May 1991, ESA Document WPP-025, Volume 1. Also published in Special Issue on AI in Space, Future Generation Computer Systems, 7, 413-437.

Grant, T. J. (1992). Integrating Payload Design, Planning and Control in the Dutch Utilisation Centre, Proceedings, 2nd International Symposium on Ground Data Systems for Space Mission Operations (SpaceOps'92), Pasadena, California, USA, 16-20 November 1992, JPL Publication 93-5, 237-242.

Grant, T. J. (1993). Beyond Objects: An Agent-Based Simulation Tool, Proceedings, 1993 European Simulation Symposium (ESS'93), Delft, The Netherlands, 25-28 October 1993, 665-670.

Grant, T. J. (1993). Why Two Heads are Better than One, 1993 Promovendi-dag, Rijksuniversiteit Limburg, Maastricht, The Netherlands, 9 December 1993.

Grant, T. J., and F. H. Tusveld. (1992). Interface Control Document for DUC-Pilot Activity Scheduling and Diagnosis System, BSO/Aerospace & Systems Technical Report 2205792, Issue D, 3 August 1992.

Grant, T. J., R. J. van Meenen, and A. J. Stroobach. (1992). Recognising Objects by Cooperating Prototypes, 1992 Workshop on Cooperating Knowledge-Based Systems (CKBS'92), University of Keele, England, 24-25 September 1992.

Grant, T. J., and K. J. Poulter. (1993). European Initiative for Knowledge Representation Standardisation, Proceedings, 4th workshop, Artificial Intelligence and Knowledge-Based Systems for Space, ESTEC, Noordwijk, The Netherlands, 17-19 May 1993, 303-314.

Grant, T. J., and J. H. J. Lenting. (1993). An Arbitration Protocol for Inter-Agent Learning, 1993 Cooperating Knowledge-Based Systems (CKBS'93) workshop, Keele University, UK, 8-10 September 1993.

Gujer, J. J., and E. Jabs. (1991). Use of Spacecraft Simulators at ESOC, ESA Bulletin, 59, 40-48.

Huhns, M. N. (ed). (1987). Distributed Artificial Intelligence, Morgan Kaufman Publishers, Los Altos, California, USA.

Lesser, V. R., and D. D. Corkill. (1981). Functionally Accurate, Cooperative Distributed Systems, IEEE Transactions on Systems, Man, and Cybernetics, SMC-11, 1, 81-96. Also published in (Bond and Gasser, 1988), 295-310.

Michalski, R. S., J. G. Carbonell, and T. M. Mitchell, (eds). (1983). Machine Learning: An Artificial Intelligence Approach, Volume 1, Morgan Kaufman, San Mateo, CA, USA.

Minsky, M. (1986). The Society of Mind, Simon and Schuster, New York, USA.

Pidgeon, A. N., B. Seaton, G. Howard, and K-U. Peters. (1992). Spacecraft Autonomy Concept Validation by Simulation: Phase 2 Final Report, ESA CR(P) 3604, Issue 1, 28 August 1992.

Pronk, C. N. A., N. Koopman, and D. de Hoop. (1992). Development Concept for Dutch User Support, Paper IAF-92-0711, Proceedings, 43rd Congress, International Astronautical Federation, 28 August to 5 September 1992, Washington D.C., USA. Also available as NLR TP 92272 L, provisional issue dated 25 June 1992, Nationaal Lucht- en Ruimtevaartlaboratorium, Amsterdam, The Netherlands.

Pronk, C. N. A., F. B. Visser, and R. M. M. Sijmonsma. (1993). Preparation and Demonstration of a Support Technology Concept for In-Orbit Payload Operations, Proceedings, 3rd European In-Orbit Operations Technology Symposium, ESTEC, Noordwijk, The Netherlands, 22-24 June 1993.

Searle, J. R. (1980). Minds, Brains and Programs, The Behavioural and Brain Sciences, 3, 417-24. Also published in Boden, M. A. (ed). (1992). The Philosophy of Artificial Intelligence, Oxford Readings in Philosophy, Oxford University Press, Oxford, UK, 67-88.

Waltz, A., and J. Llinas. (1990). Multisensor Data Fusion, Artech House Inc, Boston, USA.

Widman, L. E., and K. A. Loparo. (1989). Artificial Intelligence, Simulation, and Modeling: A Critical Survey. In Widman, L. E., K. A. Loparo, and N. R. Nielsen, (eds). (1989). Artificial Intelligence, Simulation and Modeling. John Wiley & Sons, New York, USA, 1-44.

Wohl, J. G. (1981). Force Management Requirements for Air Force Tactical Command and Control, IEEE Transactions in Systems, Man, and Cybernetics, SMC-11, 618-639.

Call for Papers

NASA 1995

Goddard Conference on Space Applications of Artificial Intelligence

May 1995
NASA Goddard Space Flight Center
Greenbelt, Maryland

The Tenth Annual Goddard Conference on Space Applications of Artificial Intelligence will focus on AI research and applications relevant to space systems, space operations, and space science. Topics will include, but are not limited to:

- Knowledge-based spacecraft command & control
- Expert system management & methodologies
- Distributed knowledge-based systems
- Intelligent database management
- Fault-tolerant rule-based systems
- High Performance Computing
- Fault isolation & diagnosis
- Planning & scheduling
- Knowledge acquisition
- Robotics & telerobotics
- Neural networks
- Image analysis

Original, unpublished papers are now being solicited for the conference. Abstracts should be 300–500 words in length, and must describe work with clear AI content and applicability to space-related problems. Two copies of the abstract should be submitted by September 1, 1994 along with the author's name, affiliation, address and telephone number. Notification of tentative acceptance will be given by September 16, 1994. Papers should be no longer than 15 pages and must be submitted in camera-ready form for final acceptance by November 16, 1994.

Accepted papers will be presented formally or as poster presentations, which may include demonstrations. All accepted papers will be published in the Conference Proceedings as an official NASA document, and select papers will appear in a special issue of the international journal *Telematics and Informatics*. There will be a Conference award for Best Paper.

Please e-mail or FAX submissions if possible.

No commercial presentations will be accepted

Sponsored by NASA/GSFC

Mission Operations and
Data Systems Directorate

1995 Goddard Conference on Space Applications
of Artificial Intelligence
May 1995 — NASA/GSFC, Greenbelt, MD

❑ ***Abstracts due:*** Sept. 1, 1994
❑ ***Papers due:*** Nov. 16, 1994
❑ ***Further info:*** (301) 286-3150
❑ ***FAX:*** (301) 286-1768

❑ ***Send abstracts to:***
Walt Truszkowski
NASA/GSFC Code 522.3
Greenbelt, MD 20771
truszkow@kong.gsfc.nasa.gov

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	May 1994	Conference Publication - May 10-12, 1994

4. TITLE AND SUBTITLE

Carl F. Hostetter, Editor

5. FUNDING NUMBERS

510

6. AUTHOR(S)

1994 Goddard Conference of Space Applications of Artificial Intelligence

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS (ES)

Goddard Space Flight Center
Greenbelt, Maryland 20771

8. PEFORMING ORGANIZATION REPORT NUMBER

93B00066

9. SPONSORING / MONITORING ADGENCY NAME(S) AND ADDRESS (ES)

National Aeronautics and Space Administration
Washington, DC 20546-0001

10. SPONSORING / MONITORING ADGENCY REPORT NUMBER

NASA CP-3268

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATMENT

Unclassified - Unlimited
Subject Category 63

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This publication comprises the papers presented at the 1994 Goddard Conference on Space Applications of Artificial Intelligence held at the NASA/Goddard Space Flight Center, Greenbelt, Maryland, on May 10 - 12, 1994. The purpose of this annual conference is to provide a forum in which current research and development directed at space applications of artificial intelligence can be presented and discussed.

14. SUBJECT TERMS

Artificial Intelligence expert systems, planning, scheduling, fault diagnosis, control, knowledge representation, knowledge acquisition, neural networks, distributed systems, fuzzy logic

15. NUMBER OF PAGES

383

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL